UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Detección de señales tempranas de eventos terroristas en flujo continuo de tweets

D. Álvaro Vadillo Aguilera
Tutor: Prof. D. Pablo A. Haya Coll
Ponente: Prof. D. Germán Montoro Manrique

FEBRERO 2019

Detección de señales tempranas de eventos terroristas en flujo continuo de tweets

AUTOR: D. Álvaro Vadillo Aguilera
TUTOR: Prof. D. Pablo A. Haya Coll
PONENTE: Prof. D. Germán Montoro Manrique

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero de 2019

Resumen (castellano)

Bruselas, Paris, Londres, Toulouse, Manchester, Estambul, Barcelona, Niza, Moscú, Oslo. Estas son algunas de las muchas ciudades que han sufrido ataques terroristas desde la creación, en el año 2013, del Estado Islámico, también denominado IS o ISIS. En total, alrededor de 500 personas han perdido la vida como resultado de estos atentados en Europa y muchos miles más en África y Asia.

Uno de los problemas que más dificulta la detección de estos ataques es que su realización es llevada a cabo por células latentes o lobos solitarios que la mayor parte de las veces viven en la misma ciudad y que mediante una bomba casera o un automóvil, pueden llevarse con ellos cientos de vidas inocentes. Debido a la naturaleza de estos ataques, resulta imposible prever donde y cuando serán los atentados ¿O no…?

Se han dado varios casos en los que los terroristas han anunciado sus intenciones de provocar un atentado varias horas antes de realizarlo, por lo que la finalidad de este Trabajo Fin de Grado es la detección de posibles eventos terroristas que se anuncien con previo aviso a través de la famosa red social Twitter. Para poder llevar a cabo esta labor, ha sido necesario crear una arquitectura resultante de juntar diversas tecnologías pertenecientes a distintos ámbitos de la informática como por ejemplo el procesamiento de lenguaje natural y el big data.

La principal característica de la arquitectura es su modularidad y escalabilidad, lo que da la posibilidad de desplegar la aplicación y ejecutarla tanto en un solo ordenador como en cientos de ellos trabajando simultáneamente.

Palabras clave (castellano)

Detección de personas, ISIS, ISIL, prevención de atentados, procesamiento de lenguaje, Terrorismo, terroristas, visualización, Yihad.

Abstract

Brussels, Paris, London, Toulouse, Manchester, Istanbul, Barcelona, Nice, Moscow, Oslo. These are some of the large number of cities around the world that have been striked by terrorists attacks since the creation, in the year 2013, of the Islamic State, also know as IS or ISIS. In total, around 500 hundred people have lost their lives as a results of terrorist attacks in Europe and thousands of them in Africa and Asia.

One of the main issues that is difficulting the detection and prevention of these attacks is that they are carried out by latent cells or lone wolfs that, usually, live in the same city where they attack and that by means of a homemade bomb or a car, can kill hundreds of innocents. Due to the inherent nature of this attacks, it is impossible to foresee where and when the attacks will take place, or is it not...?

There have been several cases in which the terrorists have announced their strike plans hours before they took place, therefore, the aim of this Bachelor Thesis is the detection of possible terrorist strikes that are announced in the famous social network Twitter some time before they are actually carried out. In order to be able to accomplish that aim, it has been necessary to build up an architecture resulting from joining several IT technologies, with completelly different applications, available nowadays, such as the Natural Language Processing and Big Data.

The major characteristic of the developed architecture is its modularity and scalability, what allows the deployment of the software and execute it in one single computer or in hundreds of them working simultaneously.

Keywords

ISIS, ISIL, Jihad, language processing, people detection, prevention of attacks, Terrorism, terrorists, visualization.

Agradecimientos

A mi madre, por obligarme a estudiar desde que tengo memoria.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

El 11 de agosto de 1988 se fundó la famosa organización yihadista *Al Qaeda* bajo el mandato de su líder Osama Bin Laden. Este grupo armado perpetró numerosos atentados a lo largo de todo el mundo provocando miles de muertes a su paso y no fue hasta el 2013 cuando se produjo una ruptura interna de la organización dando lugar al ISIL.

Desde el año 2016, se han producido numerosos ataques yihadistas que tienen como objetivo principal Europa y sus habitantes. Estos ataques han sido ejecutados, en su mayoría, por los determinados *lobos solitarios*, es decir, personas que actúan solas y que aparentemente no tienen ninguna vinculación con el yihadismo por lo que no siguen ninguna regla.

La idea de realizar este Trabajo de Fin de Grado surge tras provocarse el atentado que dejó tras de sí 22 muertos y 59 heridos en el Manchester Arena durante un concierto al que en su mayoría asistieron adolescentes. Este atentado tuvo una particularidad que lo diferenció del resto. El yihadista Salman Abedi, publico en Twitter (véase Figura 1-1) sus intenciones de atentar 4 horas antes de que se produjera.

Figura 1-1 Tweet publicado por el autor del atentado

Como se puede observar, el tweet fue publicado a las 18:32 PM anunciando un ataque bajo el mensaje *"¿Habéis olvidado nuestras amenazas? Se trata solo de terror"* junto con los hashtags *#ISLAMICSTATE #manchesterarena*.

Existen herramientas de rastreo basada en redes sociales similares, como por ejemplo Risk-Track [1], un proyecto europeo participado por la Universidad Autónoma y co-financiado por el Justice Programme de la Unión Europea (2014-2020), lo cual aporta una idea del interés que suscita este tipo de tecnología. Esta herramienta está orientada a la valoración del riego de radicalización , mientras que el presente TFG se centra en la detección de posible eventos de naturaleza terrorista.

1.2 Objetivos

El objetivo de este Trabajo Fin de Grado es diseñar una arquitectura que, haciendo uso de un procesador de lenguaje, sea capaz de detectar si un determinado tweet podría tratarse o no de una amenaza terrorista. Para lograrlo es necesario combinar varias tecnologías de tal modo que trabajen de forma síncrona y en paralelo.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Tecnologías relacionadas**

  En este segundo capítulo, se explica detalladamente todas las tecnologías que componen el proyecto y cuál es su función dentro del mismo haciendo especial hincapié en el motivo por el cual han sido seleccionadas esas y no otras tecnologías para la realización del trabajo.

- **Diseño**

  En este capítulo 3, se explica cuál es la arquitectura que se ha seleccionado y como se realiza la integración entre las distintas herramientas que la componen desde un punto de vista de alto nivel.

- **Desarrollo**

  En el apartado de desarrollo, se explica cómo es la integración entre las herramientas, pero en este caso a bajo nivel. También es en este punto en el que se explica cómo se lleva a cabo el despliegue y configuración de las aplicaciones necesarias para el funcionamiento del proyecto.

- **Integración, pruebas y resultados**

  En el capítulo 5, se detallan los resultados obtenidos tras el despliegue de la aplicación mostrando los gráficos pertinentes.

- **Conclusiones y trabajo futuro**

  Por último, en el capítulo 6, se detallan las conclusiones a las que se ha llegado tras realizar el análisis de los resultados y se proponen posibles mejoras que se deberían realizar para aumentar la eficacia de la aplicación.

2 Tecnologías relacionadas

En este apartado, se detallarán las diferentes plataformas, herramientas y lenguajes de programación utilizados para el desarrollo del proyecto. Del mismo modo, se expondrán las diferencias que pueda haber con otras herramientas ya existentes dentro del mismo sector.

Los datos se han obtenido a partir de la API de la plataforma Twitter. Las principales herramientas informáticas utilizadas en este proyecto han sido las colas Kafka para el envío de información entre los programas, la biblioteca de procesamiento de lenguaje natural CoreNLP y la tecnología ELK stack que se compone de tres proyectos de código abierto que son: ElasticSearch, Logstash y Kibana para el almacenamiento de los datos, el análisis y la visualización de los resultados.

2.1 Elección de la red social

Antes de empezar a desarrollar nuestra herramienta para detectar posibles eventos terroristas, era necesario investigar las principales redes sociales que utiliza el Estado Islámico (EI) para extender su propaganda a todo el mundo. Las redes sociales que se analizaron son: WhatsApp, Telegram, Twitter, Facebook, Youtube e Instagram.

- **WhatsApp y Telegram**

  En un principio se pensó en utilizar Telegram o WhatsApp como red social para la ingesta de la información, pero se tuvieron que descartar debido a que el acceso a los canales con material yihadista suele estar restringido. El principal canal es el denominado "Nashir", *distribuidor* en árabe. Este canal cuenta con más de 4.500 suscriptores.

- **Facebook e Instagram**

  Aunque sí que se realizan posts de terroristas en estas plataformas, estas no son las preferidas por el EI debido que su expansión se encuentra limitada a los seguidores de un usuario y su impacto es mucho menor que en las demás redes sociales.

- **YouTube**

  La red social YouTube también ha sido utilizada en numerosas ocasiones por grupos terroristas. El problema de esta red social es que es utilizada para proclamar la autoría de los atentados, y no para publicar intenciones por parte de sus miembros por lo que también la hemos descartado.

- **Twitter**

  Si hablamos de difusión, Twitter es la red social por excelencia. No importa si apenas tienes seguidores, mediante el uso de los hashtags, puedes conseguir que tu contenido sea visualizado por millones de personas alrededor del mundo. Esta red social es

perfecta para los denominados *lobos solitarios*, ya que puedes escribir tus pensamientos independientemente de lo importante que seas dentro del estado islámico. Por ejemplo, en los canales de Telegram, solo el creador del canal puede publicar.

El impacto en esta red social es tan grande, que consiguen hacer llegar la información a los demás usuarios mediante la realización de *trending topics*. Un ejemplo de la magnitud fue la decapitación del periodista estadounidense Steven Sotloff, la cual se había tuiteado más de 750 veces en apenas 6 horas, y unas 1700 cuentas enviaron más de 3000 mensajes tanto en árabe como en ingles apoyando la ejecución. Dichos mensajes, llegaron a más de 2,5 millones de usuarios a lo largo de todo el mundo.
Aunque Twitter suele bloquear las cuentas con propaganda terrorista, los miembros del Estado Islámico se crean más cuentas porque como ya hemos comentado anteriormente, no es importante el número de seguidores de la cuenta, sino el "ruido" que se puede hacer con ella en la red. [2]

2.2 Uso de Twitter como ingesta de información

Twitter es un servicio de microblogging en el que los usuarios pueden compartir información mediante mensajes de 280 caracteres (inicialmente los mensajes constaban sólo de 140 caracteres). Sus más de 320 millones de usuarios hacen que sea la red social más adecuada para compartir información públicamente. Por este motivo se ha elegido esta red social como fuente de datos para la realización del Trabajo de Fin de Grado. [3]
A continuación, se va a definir unos términos utilizados en Twitter y a continuación se explicará cómo funciona la API de Twitter que se ha utilizado.

2.2.1 Términos de Twitter

- Un ***tweet*** es un mensaje de hasta un máximo de 280 caracteres que un usuario comparte en la plataforma. Este mensaje puede constar de elementos como menciones a otros usuarios, imágenes, videos, gifs, enlaces a webs externas, hashtags, etc.

- El ***feed*** es el tablón donde se muestran todos los tweets que ha publicado un determinado usuario.

- El ***timeline*** es la lista de tweets que publican los usuarios a los que sigue otro determinado usuario. Consta de tweets, respuestas a tweets y retweets.

- Los ***followers*** de un usuario, son las personas que le siguen de tal modo que cada vez que publica un tweet, este aparece en el timeline de cada uno de ellos.

- El ***retweet*** es el modo de compartir o recomendar tweets. Cuando un usuario da retweet a un tweet, este aparecerá en su feed y en el timeline de sus seguidores.

- Un ***hashtag*** es la representación de un tema en Twitter. Se escribe comenzando con el símbolo almohadilla # y sirve para que los usuarios puedan buscar tweets relacionados con dicho tema.

2.2.2 API de Twitter

Antes de comenzar a hablar sobre la API de Twitter, es conveniente definir qué es y para que las utilizan los programadores. Una API, del inglés *Application Programming Interface*, es básicamente un *puente* o punto de conexión entre los informáticos y las aplicaciones que quieren explotar. Dicho de otro modo, la API de Twitter es el conjunto de comandos, funciones y protocolos que son proporcionados al programador para que los use como vea conveniente de cara a desarrollar un producto.

Twitter tiene varias modalidades de API siendo las más utilizadas:

- ***Search Tweets***

  Esta API da la posibilidad de realizar búsquedas de tweets entre los publicados en los últimos 7 días, 30 días, o de manera ilimitada. Tiene un ratio de respuesta de unos 500 resultados cada 5 minutos.

- ***Streaming Tweets***

  Esta API permite obtener tweets en tiempo real. También se le puede aplicar filtros, como por ejemplo por idioma del tweet o por las palabras que contiene. Los tweets son devueltos en formato JSON.

La herramienta que se ha utilizado para realizar la ingesta de información es Twitter4J, la cual es una biblioteca Java no oficial especialmente desarrollada para ser usada con la API de Twitter.

Para poder utilizar esta librería es necesario crear unas credenciales desde la página oficial de Twitter de tal modo que nos podamos conectar a la API. El proceso para generar estas credenciales se explica en el apartado *4.1.1 Codificación de la ingesta de información* y se adjunta también un ejemplo de respuesta JSON en el anexo *B. Ejemplo de una respuesta de Twitter*.

2.3 Apache Kafka

2.3.1 ¿Qué es Apache Kafka?

Apache Kafka es un sistema distribuido, replicado y particionado de almacenamiento basado en el modelo de colas publicador/suscriptor. Posee una tasa de lectura y escritura muy elevada, por lo que es una herramienta idónea para gestionar flujos de información entre sistemas.

Este sistema se ha utilizado en el proyecto gracias a la enorme escalabilidad que provee, es decir, gracias a Kafka, podríamos desplegar el proyecto tanto en un solo dispositivo como en cientos de ellos según sea necesario.

2.3.2 Términos de Kafka

A continuación, se mencionan y se detallan algunos conceptos fundamentales de Apache Kafka.

- ***Topic***: Es el elemento básico de Kafka. Todos los mensajes son clasificados y almacenados dentro de un *topic*.

- ***Producer***: Son los clientes que producen los mensajes en Kafka. Estos publicadores, pueden escribir los mensajes en uno o varios *topics*.

- ***Consumer***: Los consumidores son los que reciben los mensajes que han enviado los publicadores, al igual que estos, pueden estar suscritos a uno o varios *topics* al mismo tiempo.

- ***Broker***: Los *brokers* son los nodos de Kafka. Juntos forman el *clúster*.

En la figura 2-1 se puede ver el modelo publicador-consumidor y como están distribuidos los anteriores componentes dentro de su arquitectura.

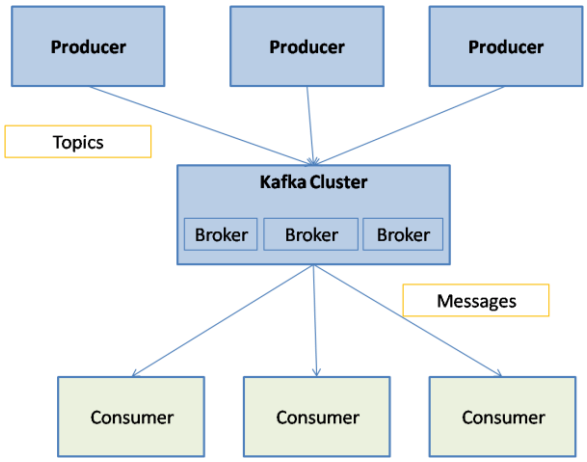

Figura 2-1 Modelo Publicador-Consumidor de Kafka

2.3.3 Sistema de colas

Una cola de mensajes, en ingles *queue*, es el elemento de Kafka por el cual los mensajes son transmitidos. Su objetivo es garantizar la entrega de los mensajes. Si el destinatario no está disponible cuando los mensajes llegan a su destino, estos quedarán guardados siguiendo un modelo FIFO hasta que puedan ser entregados correctamente.

Hay dos tipos de sistemas de colas:

- **Modelo Punto a Punto**

  En este caso el mensaje es enviado por un publicador y es recibido por un solo suscriptor.

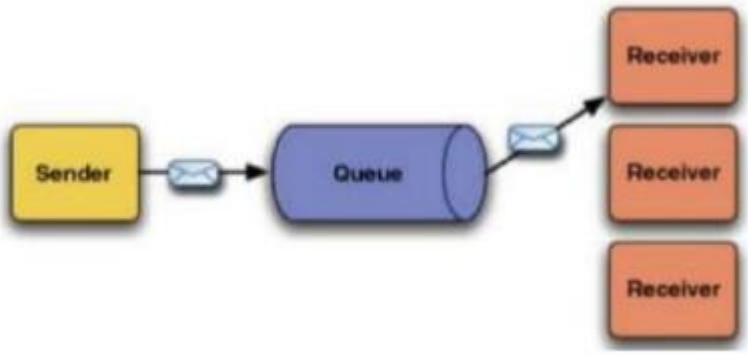

Figura 2-2 Modelo Punto a Punto

- **Modelo Publicador / Suscriptor**

  En este modelo, el mensaje no solo es recibido por un suscriptor, sino que es recibido por N suscriptores.

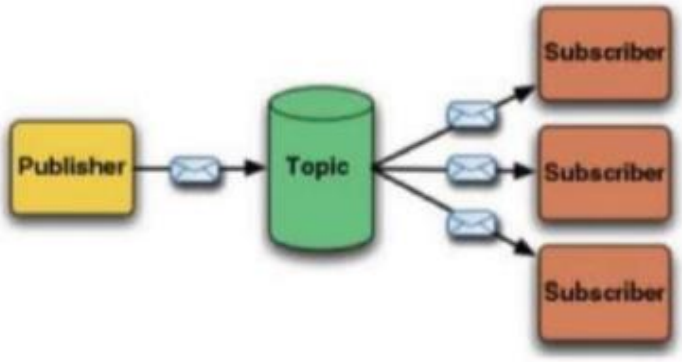

Figura 2-3 Modelo Publicador/Subscriptor

2.3.4 Topics

Como hemos dicho anteriormente, un *topic* es el elemento de Kafka donde son publicados los mensajes y enviados a cero, uno o múltiples suscriptores. En la figura 2-4, se puede observar como cada *topic* mantiene la información en un log particionado.

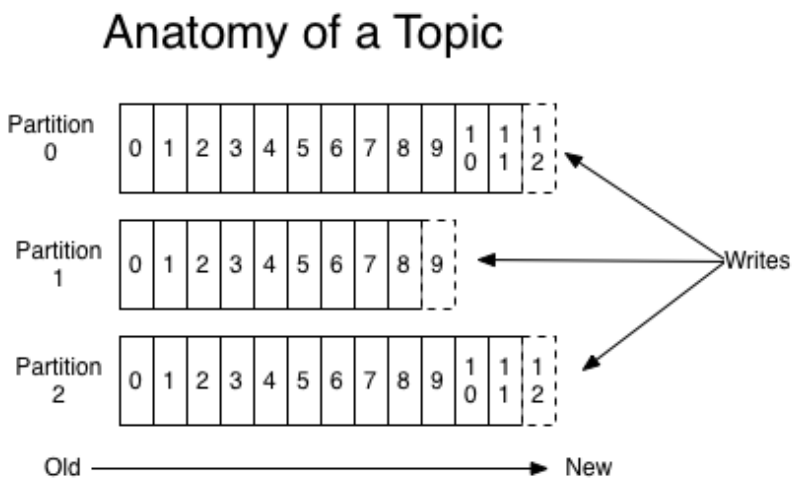

Figura 2-4 Anatomía de los topics

Cada partición almacena una secuencia de mensajes que no puede ser alterada. Según van llegando los mensajes se van añadiendo a las particiones donde se les asigna un número secuencial denominado offset utilizado para identificar cada mensaje en su correspondiente partición.

Todos los mensajes estarán almacenados y disponibles para su consumo durante una ventana de tiempo que puede ser configurada. Una vez superado este tiempo, la cola eliminará los mensajes.

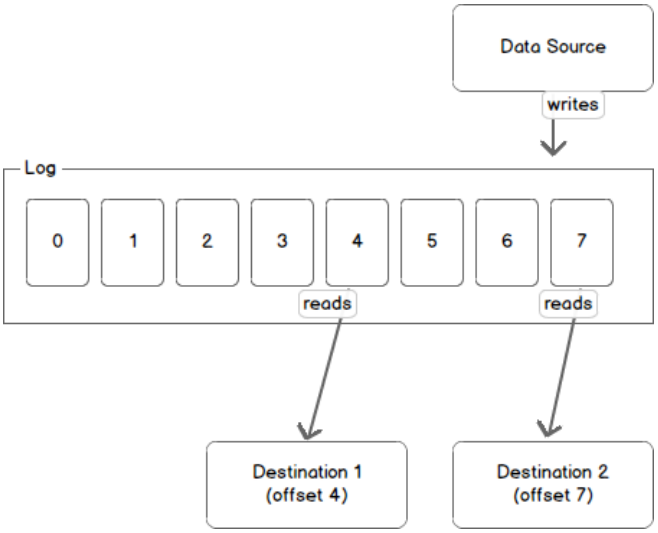

Figura 2-5 Escritura y lectura del topic

2.3.5 Apache ZooKeeper

Apache ZooKeeper es un servicio especialmente orientado a la coordinación de procesos distribuidos. Con ZooKeeper se pueden almacenar determinadas configuraciones de forma distribuida por lo que es tolerante a fallos y posee alta disponibilidad. En ocasiones es utilizado para restaurar estados tras producirse una fallo o caída del servicio.

ZooKeeper es necesario para el funcionamiento de Kafka y sirve como coordinador entre los consumidores y los *brokers*. También mantiene un registro con los consumidores que existen, así como los *topics* creados. De este modo, si un cliente antiguo se reconecta, ZooKeeper se encarga de realizar la conexión con Kafka y enviarle los mensajes que le faltan.

2.4 Procesamiento de lenguaje

Actualmente, existen numerosas herramientas dedicadas específicamente al procesamiento de la información, más concretamente al procesamiento de lenguaje. Este procesamiento se puede realizar a distintos niveles en función de la finalidad que queramos abordar, de tal modo que distinguimos los siguientes tipos:

- **Nivel fónico:**

  Estudia las unidades más pequeñas e indivisibles de la lengua. Estas unidades poseen solo significante, es decir, que no poseen significado hasta que no son combinadas con otras unidades. Un **fonema** es la imagen abstracta de un sonido. Este nivel es importante debido a que mismos fonemas pueden variar su significado dependiendo de se digan.

- **Nivel morfológico**

  El nivel morfológico es el segundo nivel. Estudia la composición de las palabras, sus partes y sus clases mediante el análisis de los **morfemas**. Los morfemas resultan de la unión de varios fonemas. [4]

- **Nivel sintáctico:**

  En el tercer nivel o nivel sintáctico, se estudian las combinaciones de palabras que son utilizadas para expresar una idea mediante el uso de estructuras complejas. De este modo, un enunciado es un conjunto de palabras o morfemas combinados. La disciplina que estudia el nivel sintáctico de las oraciones se denomina Sintaxis.

- **Nivel semántico:**

  Este nivel estudia el significado de las palabras y la relación entre el significante y el significado. Su unidad mínima es el sema (unidad mínima de significado)

- **Nivel pragmático:**

  El nivel pragmático hace referencia al uso social del lenguaje de manera interactiva. En otras palabras, estudia el significado de un texto en función del contexto.

El procesador de lenguaje que se ha elegido para la realización de este proyecto ha sido CoreNLP, ya que, aunque en un primer momento, el trabajo consiste solo en analizar frases escritas en inglés, CoreNLP tiene un módulo para el procesamiento del lenguaje escrito en **árabe** que podría ser utilizado en un futuro. En el siguiente apartado, se detallan algunas tecnologías similares con sus principales caracteristicas.

2.5 Elección del procesador del lenguaje

El principal problema que se ha tenido que afrontar a la hora de la realización de este trabajo, ha sido **la ambigüedad** en las frases. Para poder hacer frente a este problema, es necesario definir una cantidad bastante grande de reglas y estructuras lingüísticas. Aun así, estas herramientas son capaces de tener la ambigüedad en cuenta, aunque esto suponga un mayor tiempo de procesamiento.

A continuación, se muestran las herramientas que se investigaron para la realización del proyecto.

2.5.1 NLTK

NLTK son las siglas de *Natural Language Toolkit*¸ una librería desarrollada en Python para la investigación del procesamiento natural. NLTK posee más de 50 recursos léxicos como WordNet. Esta herramienta fue descartada por no soportar el árabe.

NLTK consta de los siguientes módulos:

- Tokenizador
- Análisis sintáctico
- Reconocimiento e identificación de entidades
- Etiquetado morfosintáctico

2.5.2 Gate

Gate son las siglas de *General Architecture for Text Engineering*, una herramienta creada por la Universidad de Sheffield que ejecuta secuencialmente una serie de módulos de procesamiento a nivel lingüístico para la realización de tareas de todo tipo. Sus módulos básicos son:

- Tokenizador
- Análisis sintáctico
- Reconocimiento e identificación de entidades
- Etiquetado morfosintáctico
- Segmentación de oraciones
- Resolución de la conferencia

Esta herramienta fue descartada por no proveer un analizador de dependencias ni etiquetado POS.

2.5.3 Spacy

Al igual que las anteriores herramientas, Spacy es una biblioteca para el procesamiento del lenguaje. Está escrita en Python y Cython, y ha sido desarrollada por Matthew Honnibal bajo la licencia del Instituto Tecnológico de Massachusetts (MIT). Ofrece modelos estadísticos basados en redes neuronales con integración para más de 33 idiomas distintos y sus principales características son:

- Reconocimiento e identificación de entidades
- Tokenizador
- Modelos estadísticos entrenados
- Etiquetado POS (*Part of speech)*
- Análisis sintáctico
- Análisis de dependencias
- Etiquetado morfosintáctico
- Segmentación de oraciones

Aunque es considerada la herramienta más rápida del mercado, se ha decidido descartarla debido a que no posee modelos estadísticos para el árabe. Además de que está escrita en Python y era más conveniente una herramienta escrita en Java, al igual que el resto de la arquitectura.

2.5.4 CoreNLP

CoreNLP ha sido desarrollado por la universidad de Standford y proporciona diferentes herramientas para el procesado de texto. Soporta varios lenguajes entre los cuales se encuentra el árabe, idioma de especial interés para la realización de este trabajo.

Está escrito en Java, aunque también existen implementación en otros lenguajes de programación. Es fácil de usar e integra varios módulos como:

- Tokenizador
- Reconocimiento e identificación de entidades
- Etiquetado morfosintáctico
- Análisis de dependencias
- Análisis sintáctico
- Etiquetado POS (*Part of speech)*

A continuación, se muestra, a modo de ejemplo, como es analizada una frase en español: *El rápido zorro marrón saltó sobre el perro perezoso*

Etiquetado POS (Part of speech):

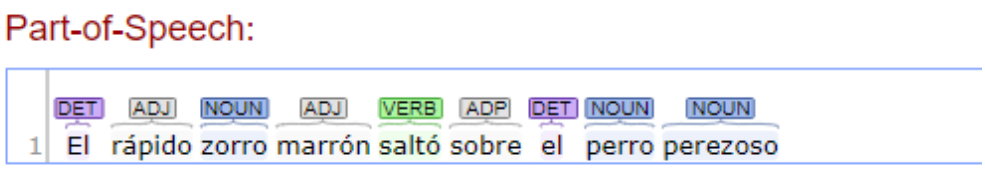

Figura 2-6 Ejemplo de etiquetado POS

Como se puede observar en la figura 2-6, se detecta correctamente los determinantes, adjetivos, verbos y nombres de la frase.

Análisis de dependencias:

En la figura 2-7, se puede observar un ejemplo de analisis de dependencias realizado con la frase descrita anteriormente.

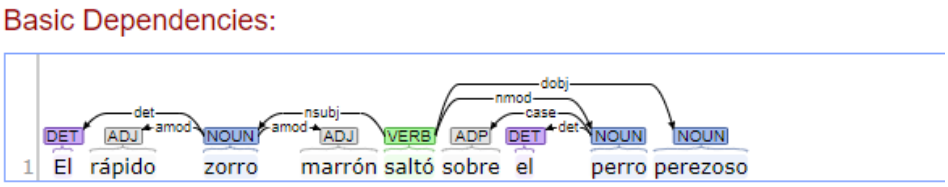

Figura 2-7 Ejemplo de análisis de dependencias

En este segundo nivel, ya son categorizadas las relaciones entre las palabras de tal modo que obtendriamos información basada en pares de palabras como por ejemplo: *det(El, zorro); nsubj(zorro, saltó) o case(sobre, perro)*

Ha sido elegida esta herramienta entre todas las anteriores por varias razones, la primera es que está escrita en Java, por lo que nos da la facilidad de importar las dependencias con

Maven y crear paquetes JAR listos para ejecutar en tantos ordenadores como sea necesario. La segunda razón es su fácil uso, ya que con una simple línea de código podemos habilitar o deshabilitar módulos según se quiera.

2.6 ELK Stack

ELK es el acrónimo de tres proyectos *open source* los cuales son **ElasticSearch, Logstash y Kibana.** ElasticSearch es un motor de búsqueda y análisis. Logstash es una herramienta para el procesamiento de datos del lado del servidor que ingiere datos de múltiples fuentes simultáneamente, las transforma y se las envía a un motor de búsqueda como es ElasticSearch. Kibana permite a los usuarios visualizar los datos vía web con tablas y gráficos en tiempo real.

Figura 2-8 Elementos de ELK

En nuestro caso, los ficheros de logs serán los mensajes recibidos mediante las colas Kafka. Esto es posible gracias a que Logstash tiene un conector con Kafka.

2.6.1 Elastic Search

ElasticSearch es un motor de búsqueda desarrollado en código abierto que posee una gran escalabilidad. Permite almacenar, buscar y analizar conjuntos muy grandes de datos en tiempo casi real y a gran velocidad. Su diferencia con la búsqueda por texto convencional reside en la utilización de un Lenguaje de Dominio Especifico, en ingles *Domain Specific Languague*. Un lenguaje de dominio especifico es un lenguaje de programación especialmente diseñado para resolver un problema en particular. En este caso, ElasticSearch utiliza *Query DSL* para realizar consultas a los documentos que tiene indexados.

Está basado en los siguientes conceptos:

- **Index:** Es una colección de documentos con características similares.
- **Clúster:** Conjunto de uno o más nodos que, de manera distribuida e indexada, mantienen toda la información,

- **Nodo:** Elemento de clúster que almacena la información, la busca y la indexa.

- **Replicas:** Son utilizadas cuando la información indexada no puede ser soportada por una sola máquina. Mediante el *sharding*, podemos dividir la información en distintos subconjuntos ofreciéndonos la posibilidad de escalar horizontalmente nuestra herramienta. Esto también nos da la posibilidad de realizar las tareas de forma paralela utilizando múltiples maquinas. ElasticSearch es tolerante a fallos por lo que puede seguir trabajando en caso de que ocurra algún fallo en alguna máquina.

Entre todas las características de ElasticSearch, hay que destacar sobre todo que se trata de una plataforma de búsqueda en tiempo casi real. El retraso desde que el índice se actualiza o se borra hasta que lo vemos reflejado es de un segundo. Otra característica muy importante es la facilidad para almacenar y recuperar texto.

Otra característica a tener en cuenta es su API. ElasticSearch proporciona una API REST para interactuar directamente con el clúster.

2.6.2 Logstash

Logstash es una herramienta que se basa en una arquitectura tipo *pipeline* parecida a Kafka. Las arquitecturas *pipeline* se caracterizan por tener tres componentes claramente diferenciados. Por un lado, está la entrada o *input*, por otro lado, está el filtro donde se realiza la transformación de los datos, y finalmente esta la salida u *output*.

Logstash permite unificar de manera dinámica los datos provenientes de diferentes fuentes y normalizar dichos datos a un único modelo.

Así, las principales funcionalidades de Logstash son las siguientes:

- Procesamiento de la información escalable horizontalmente con una gran cohesión con ElasticSearch y Kibana.

- Arquitectura de tuberías conectables.

- Numerosos plugins de código abierto desarrollados por una comunidad amplia y en crecimiento

2.6.3 Kibana

Kibana es una herramienta de análisis y visualización de datos. En este caso, está diseñada para trabajar conjuntamente con ElasticSearch mediante el modelo ELK visto anteriormente. Permite representar los datos almacenados en ElasticSearch en una gran cantidad de gráficos y tablas distintos.

Ofrece una capa de personalización muy potente sobre Logstash, pero también permite crear visualizaciones personalizadas con multitud de características. Todos los dashboards,

son creados mediante su interfaz amigable accesible a través de un navegador sin que sea necesario el uso de la programación [5]

2.7 Visualización de los datos

Actualmente se pueden encontrar numerosos motores de visualización orientados al análisis de los datos tanto almacenados durante un periodo de tiempo como en tiempo real. Las principales herramientas que se han encontrado han sido:

2.7.1 Cyfe

Cyfe es una herramienta especialmente diseñada para la visualización de datos provenientes de redes sociales. Es capaz de construir diferentes cuadros de mando usando como ingesta de información un gran número de servicios como son Facebook, Amazon, MailChimp, WordPress o Twitter (véase figura 2-9). Su principal característica es la capacidad de juntar toda la información independientemente del medio del cual provenga. Así es posible obtener de un simple vistazo toda la información de nuestras redes sociales como por ejemplo el número de seguidores, lo que están compartiendo o visualizar cualquier dato que pueda ser extraído de dichas redes. Esta herramienta fue descartada para la utilización de este proyecto porque no se visualizan los datos obtenidos directamente de Twitter, sino que se realiza una especie de filtrado previo.

Figura 2-9 Ejemplo de dashboard en Cyfe

2.7.2 Grafana

Grafana es una herramienta de código abierto que generalmente está integrada con bases de datos como Graphite o InfluxDB aunque también posee conectores para integrarla con ElasticSearch. Con Grafana, la creación de cuadros de mando es sencilla y se realiza de forma rápida.

La principal diferencia con Kibana es que Grafana está más orientada a la representación de datos como la CPU de un ordenador o el número de paquetes enviados a través de la red. Su principal carencia es que no permite la búsqueda o exploración de los datos, por lo que se ha descartado a favor de utilizar Kibana. En la figura 2-10 se muestra un ejemplo de un *dashboard* en Grafana.

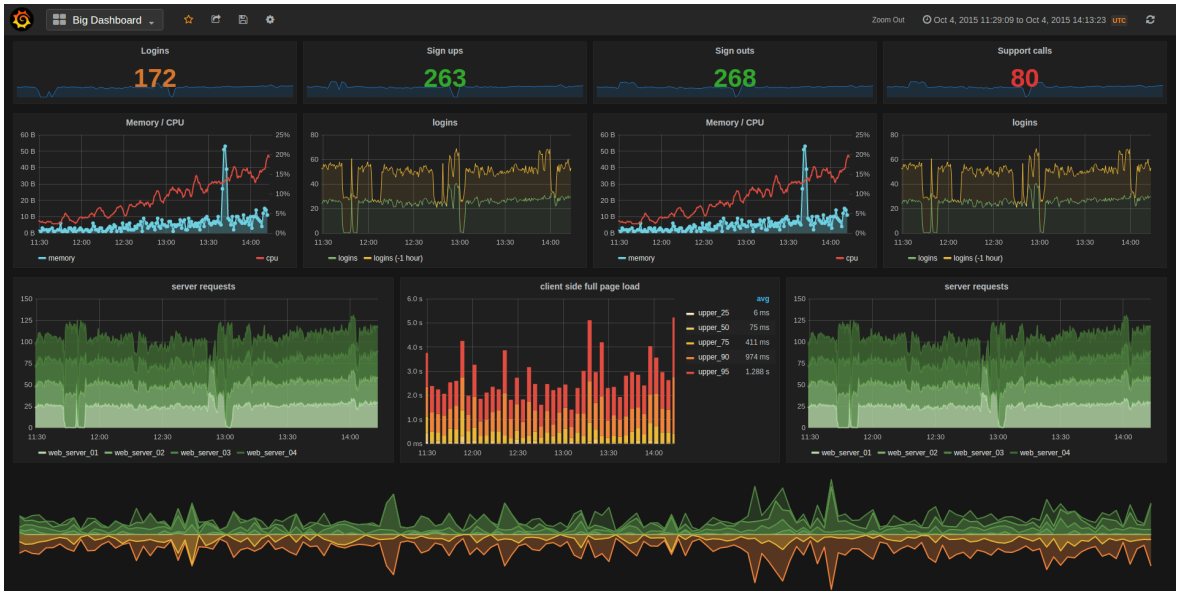

Figura 2-10 Ejemplo de dashboard en Grafana

2.7.3 Kibana

La representación de los datos en Kibana es bastante similar a la de Grafana, sin embargo, fue elegida entre las demás debido a su facilidad a la hora de crear representaciones de datos y también por su fuerte conexión con Apache Kafka mediante el uso de Logstash.

El despliegue de Kibana es muy sencillo como veremos en el apartado 4, ya que con una simple línea de código podemos conectar Kibana con los demás elementos de ELK y desplegar un servidor web donde visualizaremos los datos. En la figura 2-11 se muestra un ejemplo de un *dashboard* en Kibana.

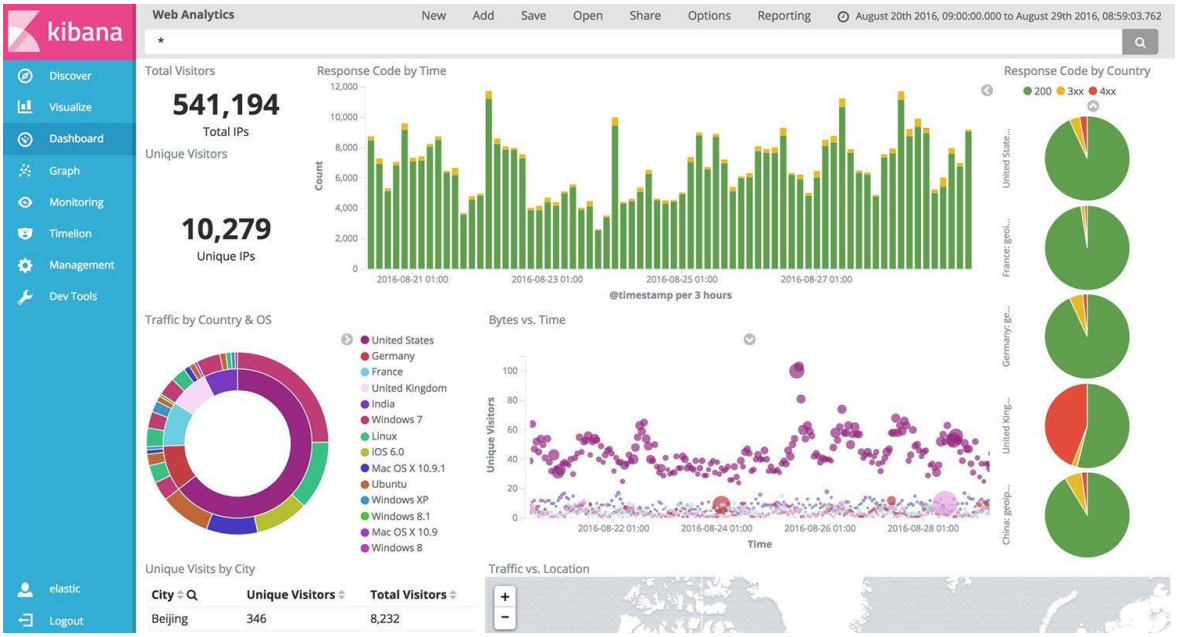

Figura 2-11 Ejemplo de dashboard en Kibana

3 Diseño

En este apartado de la memoria se va proceder a explicar el diseño que se ha decidido implementar para los dos siguientes componentes del proyecto. Por un lado, el diseño de toda la arquitectura software con sus componentes, y por el otro, el diseño del procesador de lenguaje para entender los pasos que sigue a la hora de categorizar un texto.

- **Diseño de la arquitectura**

  La principal cualidad de la arquitectura es su modularidad y desacople. Gracias a las tecnologías utilizadas, es posible crear una arquitectura totalmente escalable que puede ser desplegada en tantos nodos como se desee.

- **Diseño del procesador de lenguaje**

  La finalidad de este diseño es la creación de un software que analice textos mediante un proceso denominado análisis de dependencias. En el apartado 3-2 se explica más en profundidad como es el proceso que lleva a cabo y cuáles son sus fases.

- **Diseño de la obtención de datos**

  En este apartado se especificará como ha de ser el formato de la ingesta de los datos y que campos es necesario que posea para posteriormente pueda ser representado visualmente.

- **Diseño del almacenamiento de la información**

  En este ultimo apartado se explicarán las características que tienen que poseer los motores de procesamiento para que el proceso de almacenamiento e indexado sea óptimo.

3.1 Diseño de la arquitectura

Una vez explicados las diferentes herramientas que componen nuestro programa, nos vamos a centrar en describir cómo se integran entre sí de tal modo que funcionen de manera simultánea. En la figura 3-1 se muestra el diseño de la arquitectura escogida.

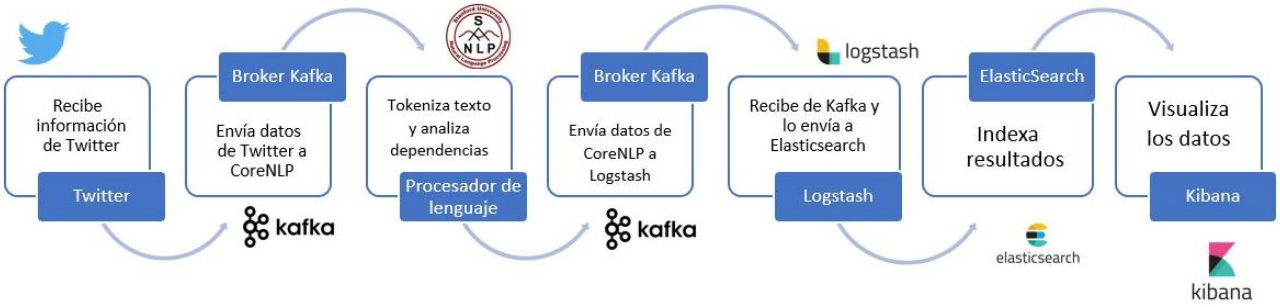

Figura 3-1 Diseño de la arquitectura

Aunque no sea visible a simple vista, la arquitectura se podría dividir en 4 subprogramas diferenciados basados en su funcionalidad. El primero de ellos, encargado de la ingesta de información, el segundo de ellos, encargado del procesamiento de la información, el tercero de ellos es el encargado de mostrar dicha información de manera visual para el usuario final y el cuarto es el correspondiente a Kafka, el cual se ejecuta de modo transversal a los demás.

Aunque en el diagrama se vea un único nodo por cada componente, hay que destacar que esto no implica que solo pueda ser ejecutado en un único host sino todo lo contrario, cualquier herramienta utilizada en esta arquitectura, puede ser ejecutada en tantos nodos como se desee con la finalidad de realizar el trabajo de manera paralela.

3.2 Diseño del analizador de dependencias

En cuanto al analizador de dependencias, su función es etiquetar las palabras de texto que se quiere analizar y clasificar su dependencia con las demás palabras del texto. Para llevar a cabo este proceso hay que realizar una serie de pasos. Estos pasos se detallan a continuación en la figura 3-2:

Figura 3-2 Diseño del analizador de dependencias

- **Texto a analizar:** Es el texto que obtenemos de la cola kafka que viene de Twitter. Este texto se corresponde con un tweet que ha pasado el primer nivel de filtro de la aplicación.

- **Segmentador:** El segundo paso del procesador de lenguaje es el denominado Segmentador, en este apartado, el texto que se va a analizar es dividido en partes más pequeñas con la finalidad de paralelizar el trabajo lo máximo posible.

- **Tokenizador:** En este apartado, se lleva a cabo el proceso de dividir las oraciones en *tokens*. Estos tokens son las palabras del texto inicial.

- **Etiquetador POS (Part Of Speech):** También denominado analizador morfosintáctico, es el encargado de categorizar gramaticalmente los *tokens* o palabras obtenidos mediante el proceso de tokenización. El etiquetado morfológico es más complejo de lo que parece a primera vista ya que, existen palabras que pueden

pertenecer a distintos grupos gramaticales. Por ejemplo, la palabra en ingles *attack* puede referirse tanto al verbo atacar como al sustantivo ataque.

- **Analizador de dependencias:** Es el último paso y es el encargado de crear pares de palabras en función de su categoría gramatical, de tal modo que es capaz de detectar objetos directos a partir de un verbo transitivo y un sustantivo.

3.3 Diseño de la obtención de datos

La función de este la obtención de datos es, como su nombre indica, la recolección de información a través de un canal determinado. En este caso, la red social escogida a sido Twitter debido a las características explicadas en el punto 2.2. Sin embargo, es posible obtener la información de canales adicionales como pueden ser otras redes sociales. Para que sea posible la conexión con CoreNLP, es necesario que dicha fuente de información provea los datos en un formato que pueda ser convertido a cadenas. Esto es debido a que las colas Kafka están configuradas para recibir datos del tipo *String*. Aparte de esta especificación, es necesario crear un JSON con los campos que utiliza ELK para la representación de las gráficas.

3.4 Diseño del almacenamiento de la información

Para la realización de este proyecto, se ha utilizado ElasticSearch como motor de búsqueda debido a que es una base de datos no relacional, también denominada NoSQL. La principal característica de las bases de datos no relacionales es que están específicamente diseñadas para modelos de datos muy específicos con esquemas muy flexibles. Entre sus puntos fuertes respecto a las bases de datos relacionales se encuentran su facilidad de desarrollo, su funcionalidad, su alto rendimiento y su gran robustez gracias a la arquitectura cluster que posee como ya se ha descrito en el apartado 2.6. [6]

Otro ejemplo de motor de búsqueda basado en bases de datos no relacionales es Solr, este motor también está basado en Lucene al igual que ElasticSearch. Sin embargo, fue escogido ElasticSearch a favor de utilizar ELK

3.5 Filtro de los mensajes

Como ya se ha comentado en el apartado 2.1.2, Twitter consta de una API desde la cual se pueden realizar acciones como publicar tweets, seguir a usuarios, compartir información, etc. Sin embargo, para este proyecto, solo es necesaria la funcionalidad de obtención de tweets. Dicha funcionalidad es configurable de tal modo que podemos solo obtener tweets que cumplan una regla que nosotros hemos establecido.

Este es sin duda uno de los puntos clave del proyecto, ya que es esencial la elección de las palabras clave que deben aparecer en el tweet para que sea seleccionado.

Los mensajes se van a filtrar en dos momentos del programa, una primera criba se realizará al obtener los tweets de la API de Twitter y una segunda criba se realizará una vez

procesado el texto. Esta segunda criba será explicada en el siguiente apartado *3.1.3 Procesador del lenguaje*.

Para el primer filtrado de mensajes, se ha elegido una lista con palabras comúnmente utilizadas por el EI en sus mensajes. Esta lista puede ser modificada en todo momento. Sus elementos son los siguientes:

- Bomb
- Explosive
- Artifact
- Vest
- Infidel
- Christian
- Occidental
- Apostate
- Islamic
- Islamic State
- ISIS
- ISIL
- YIHAD
- ALLAH
- #ISIS
- #IslamicState
- #ALLAH
- #YIHAD

En esta primera criba, se evitó utilizar verbos debido a su gran cantidad de connotaciones. También se evitaron las cadenas de texto con pocas letras como "IS" o "C4" ya que aparecen en numerosos tweets como partes de una palabra, como por ejemplo "My father **is** Luis" o "enlace.com/id=f23mc5**c4**k0"

Una vez obtenidos los mensajes que pasan este filtro, son enviados al procesador de lenguaje CoreNLP mediante una cola Kafka.

4 Desarrollo

4.1 Codificación de la ingesta de información mediante Twitter

Este módulo es el punto de partida de nuestra aplicación desde el cual se empieza a obtener información mediante la API de Twitter. Está desarrollado en Java y usa la librería *Twitter4J* para la sincronización con Twitter.

Para poder obtener datos de Twitter, es necesario dar de alta una aplicación en la página web de Twitter para desarrolladores https://developer.twitter.com.

Una vez dada de alta nuestra aplicación, se nos proporcionarán unas claves de acceso y tokens que serán necesarias incluir más adelante en nuestro código.

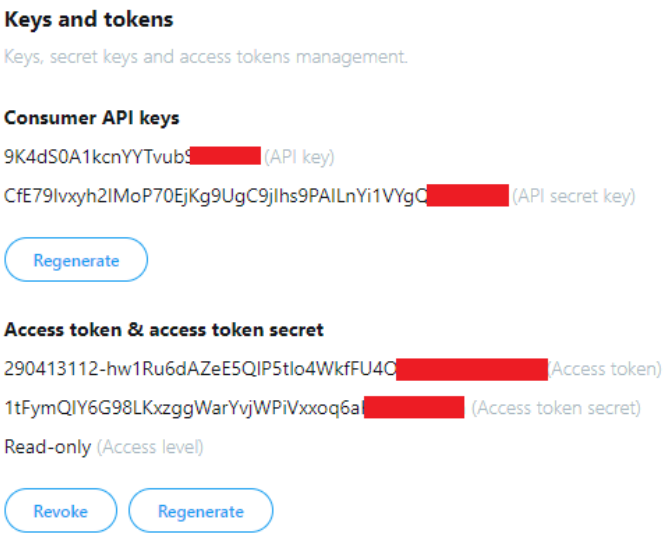

Figura 4-1 Keys y tokens de Twitter

Con estas claves, ya tendremos acceso a la API y todas sus funcionalidades, aunque en este caso solo tiene permisos *Read-only* ya que nuestra aplicación no necesita publicar tweets.

Estas claves son almacenadas en el fichero TwitterConf.java, y son accedidas desde nuestra clase principal para realizar la conexión con la API mediante el *ConfigurationBuilder*.

```
ConfigurationBuilder cb = new ConfigurationBuilder();
cb.setOAuthConsumerKey(TwitterConf.CONSUMER_KEY_KEY);
cb.setOAuthConsumerSecret(TwitterConf.CONSUMER_SECRET_KEY);
cb.setOAuthAccessToken(TwitterConf.ACCESS_TOKEN_KEY);
cb.setOAuthAccessTokenSecret(TwitterConf.ACCESS_TOKEN_SECRET_KEY);
cb.setJSONStoreEnabled(true);
```

Una vez configurado el *Builder* con unos tokens correctos, ya se puede empezar a obtener tweets.

Para ello, es necesario utilizar un *Listener* que esta implementado en la clase *TwitterStreamFactory*. Dentro de esta clase encontraremos las siguientes funciones: *onStatus, onDeletionNotice, onTrackLimitationNotice, onScrubGeo, onException, y onStallWarning.*

Usaremos *onStatus* para tratar los tweets, esta función es llamada cada vez que se recibe un tweet ejecutando el código que incluyamos en ella. Y *onException* para cerrar la conexión si se produjera algún tipo de fallo.

Como ya se explicó en el apartado 3.3, es necesario obtener tweets que contienen una palabra dentro de nuestra batería de palabras previamente diseñada. Para añadir un filtro a nuestra búsqueda de tweets hay que crearse un objeto *FilterQuery* al que le pasaremos nuestro listado de palabras potencialmente peligrosas. La codificación es la siguiente:

```
FilterQuery fq = new FilterQuery();
fq.track(words);
twitterStream.filter(fq);
```

Siendo *words* un array de Strings con las palabras explicadas en el apartado *3.2 Filtro de mensajes.*

4.2 Tratamiento de la información a través de Kafka

En este caso, el envío de la información no se podría definir como un módulo independiente si no que es la herramienta que sirve para conectar todos diferentes módulos.

Es una herramienta que se utiliza de manera trasversal a lo largo de toda nuestra arquitectura.

Lo primero que se debe hacer para poder enviar información mediante Apache Kafka, es configurar tanto Kafka como el Zookeeper.

Para ello es necesario configurar el número de particiones en *server.properties*. En este caso está configurado a 1 ya que el proyecto esta desplegado en local, pero para obtener mayor paralelismo, habría que configurar un mayor número de particiones.

Además del número de particiones, hay que configurar los puertos en los que se van a conectar los clientes (puerto 2181) así como el puerto de los servidores de Bootstrap (puerto 9092). Los servidores de Bootstrap son una lista de pares host/puerto necesario para establecer la conexión inicial con el clúster de Kafka.

Una vez establecidas estas configuraciones, es necesario ejecutar tanto el servidor de Kafka como el de ZooKeeper para poder crear *topics*.

Para desplegar el servidor de Kafka es necesario ejecutar el siguiente comando en una terminal desde dentro de la carpeta de Kafka:

```
bin/kafka-server-start.sh config/server.properties
```

Del mismo modo, para desplegar el servidor de ZooKeeper es necesario ejecutar el siguiente comando también estando posicionado en la carpeta de Kafka.

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

Con estos dos servicios ejecutándose ya es posible crear *topics* en Kafka de la siguiente manera:

```
bin/kafka-topics.sh --zookeeper localhost>:2181 --create --topic nombre_topic
--partitions 1 --replication-factor 1
```

Con este comando habremos creado un *topic* con el nombre nombre_topic. Para listar los *topics* existentes se puede utilizar el siguiente comando:

```
bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Nota: Los *topics* creados anteriormente no serán visibles hasta que se haya enviado información a través de ellos.

Una vez acabado todos estos pasos, ya tendríamos el servicio de Kafka corriendo perfectamente y solo sería necesario crear las conexiones necesarias en los módulos donde se va a utilizar. A continuación, se muestra la codificación para realizar la conexión mediante Kafka del módulo de Twitter con el módulo de CoreNLP o los submódulos de normalización.

En el módulo publicador, en este caso el módulo de Twitter, es necesario crear un objeto *Producer* desde del cual se enviará la información al módulo suscriptor que será recibido en el módulo CoreNLP usando el objeto *Consumer*.

Cuando se codifica un **productor**, es necesario establecer las siguientes propiedades:

```
Properties configProperties = new Properties();
configProperties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"localhost:9092");
configProperties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                     "org.apache.kafka.common.serialization.ByteArraySerializer");
configProperties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                      "org.apache.kafka.common.serialization.StringSerializer");
```

Cuando se codifica un **consumidor**, es necesario establecer las siguientes propiedades:

```
Properties configProperties = new Properties();
configProperties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,"localhost:9092");
configProperties.put(ConsumerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                     "org.apache.kafka.common.serialization.ByteArraySerializer");
configProperties.put(ConsumerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                      "org.apache.kafka.common.serialization.StringSerializer");
```

4.2.1 Envío de mensajes

Como ya se ha explicado, para enviar la información a través de una cola Kafka es necesario crear un *Producer* de la siguiente manera:

```
Producer<String, String> producer = new KafkaProducer<String, String>(configProperties);
```

Se puede crear con diferentes tipos de datos, pero en este caso se ha utilizado String ya que es la manera en la que es enviado el JSON que se obtiene de Twitter.

Para enviar la información a través de la cola, solo es necesario crear un *ProducerRecord* y enviarlo como se muestra a continuación:

```
ProducerRecord<String, String> record =
                  new ProducerRecord<String, String>("topic", JSON_complete.toString());
producer.send(record);
```

Siendo *topic* el nombre del *topic* por el cual queremos enviar la información y JSON_complete.toString() el tweet en cuestión.

4.2.2 Recepción de mensajes

La recepción de los mensajes a través de Kafka se realiza utilizando un consumidor. Este se crea de manera muy similar al productor que se acaba de explicar. Una vez creado el consumidor, es necesario que este se **suscriba** a un *topic* para poder empezar a leer de él.

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
consumer.subscribe(Collections.singleton("ejemplo"));
```

En este caso se ha creado un consumidor que se ha subscrito al *topic* con nombre "ejemplo" y lee datos del tipo String. Es muy importante que tanto productor como consumidor, hayan sido creados usando los mismos tipos de datos, en este caso String. En caso contrario, se provocará una excepción y se cerrará la comunicación.

Una vez subscritos al *topic*, se utilizará la clase *ConsumerRecords* para obtener los datos del mismo.

```
while (true){
        ConsumerRecords<String, String> records = consumer.poll(0);
        for (ConsumerRecord<String, String> record : records){...}
}
```

Mediante el uso del bucle *while*, el consumidor se pone a escuchar constantemente del *topic* hasta que recibe datos a través de la cola.

4.3 Procesamiento de la información

Este módulo es el que se encarga de procesar la información y es el segundo de los tres más importantes (Twitter, NLP, ELK). Al igual que el módulo de Twitter, está desarrollado en Java y posee dos conectores Kafka, uno para leer la información proveniente de Twitter, y otro para enviar los mensajes a ELK stack. La creación del productor y del consumidor se realiza del mismo modo que en el anterior apartado.

Lo primero que hay que codificar a la hora de desarrollar una aplicación que procese lenguajes es el modelo de datos que va a utilizar para el análisis de nuestros datos, así como como el conjunto de etiquetas que va a calificar.

```
String modelPath = DependencyParser.DEFAULT_MODEL;
String taggerPath = "edu/stanford/nlp/models/pos-tagger/english-
left3words/english-left3words-distsim.tagger";
```

En modelo que se ha utilizado es el predeterminado ya que se ajusta perfectamente a nuestra necesidad. En cuanto al *tagger* o etiquetador que se ha utilizado, ha sido el *english-left3words*. Este etiquetador es el recomendado por los creadores debido a que es el que tiene mayor ratio velocidad/precisión. Tiene un 96,97% de precisión frente al 97,32% del *"standard WSJ22-24 test set"* pero es mucho más rápido. [7]

Una vez establecida la configuración inicial, hay que cargar el modelo y el etiquetador. Este proceso dura unos segundos y se realiza de la siguiente manera.

```
MaxentTagger tagger = new MaxentTagger(taggerPath);
DependencyParser parser = DependencyParser.loadFromModelFile(modelPath);
```

Tras cargar el modelo y el etiquetador, ya es posible dividir las frases en *tokens*, para su posterior análisis. Para ello, hay que realizar un primer preprocesado del texto creando una lista de objetos *HasWord*. Por cada objeto de esta lista, se extrae otra lista de palabras calificadas o también denominadas *TaggedWords*. Posteriormente, hay que predecir la estructura gramatical de la palabra haciendo uso del modelo de datos cargado anteriormente. Una vez se tiene el tipo de dependencia, se comprueba si esta es un objeto directo y si es así, se añade una bandera en un campo del JSON de Twitter y se envía a través de Kafka para que sea tratado por ELK stack como se explicará en el siguiente apartado. [8]

La codificación de todo este proceso sería la siguiente:

```
DocumentPreprocessor tokenizer = new DocumentPreprocessor(new StringReader(frase));
DependencyParser parser = DependencyParser.loadFromModelFile(modelPath);
for (List<HasWord> sentence: tokenizer) {
                        List<TaggedWord> tagged = tagger.tagSentence(sentence);
                        GrammaticalStructure gs = parser.predict(tagged);
                        for (TypedDependency typed : gs.typedDependencies()) {
                            if ((typed.toString().contains("dobj")){
                                    //MODIFICACION DEL JSON
                             }
                        }
//ENVÍO POR KAFKA
}
```

4.4 Visualización de la información

La visualización de los datos se lleva a cabo gracias a Logstash, ElasticSearch y Kibana, apartado 2. Estas herramientas trabajan conjuntamente para proporcionar al usuario una interfaz fácil y sencilla desde la cual poder analizar los datos. Al tratarse de una especie de *framework* diseñado para que trabajen conjuntamente, apenas es necesario realizar modificaciones de código para conseguir su funcionamiento.

El único fichero de configuración que hay que modificar es el fichero logstash.conf, correspondiente al fichero de configuración de Logstash. En este fichero se define cual va a ser el canal por el cual se va a recibir la información y dónde va a ser enviada. El contenido del fichero es el siguiente.

```
input {
  kafka {
    bootstrap_servers => "localhost:9092"
    topics => ["topic"]
    codec => json {charset => "UTF-8"}
  }
}
output {
  elasticsearch {
    hosts => ["localhost:9200"]
    index => "elasticse"
  }
}
```

Para especificar que los datos los vamos a obtener de Kafka, solo hay que configurar los servidores Bootstrap, el *topic* del cual se va a leer la información y la codificación de los caracteres.

Por otro lado, para especificar que los datos los vamos a enviar a ElasticSearch, solo es necesario la dirección y puerto donde está corriendo el servicio y el nombre del índice que queremos crear.

Los comandos para desplegar los servicios son los siguientes.

- **Elasticsearch**: ./bin/elasticseach (Ejecutar sin ser root)
- **Kibana**: ./bin/kibana
- **Logstash**: ./logstash -f logstash.conf (Con el parámetro -f se especifica el fichero de configuración)

4.5 Construcción de ficheros JAR con Maven

Tanto para el módulo de Twitter como para el de CoreNLP, se ha utilizado la herramienta Maven. Maven es una herramienta de gestión y construcción de proyectos Java. Se ha utilizado para poder ejecutar los programas en nodos independientes sin la necesidad de disponer de un IDE desde el cual ejecutarlo. Además, no es necesario descargar e importar las librerías necesarias para el correcto funcionamiento del proyecto puesto que ya vienen definidas en el fichero *pom.xml*

Para poder generar un fichero JAR con las dependencias necesarias, hay que añadir el siguiente fragmento de código al fichero pom.xml

```
<build>
    <plugins>
        <plugin>
             <artifactId>maven-assembly-plugin</artifactId>
             <configuration>
                 <archive>
                     <manifest>
                          <mainClass>Main</mainClass>
                     </manifest>
                 </archive>
                 <descriptorRefs>
                      <descriptorRef>jar-with-dependencies</descriptorRef>
                 </descriptorRefs>
             </configuration>
        </plugin>
    </plugins>
</build>
```

Además de este código, hay que añadir todas las dependencias necesarias. Estas dependencias se encuentran en el anexo *C. Ficheros Maven*

Para compilar el proyecto y ejecutarlo solo hay que escribir los siguientes comandos:

```
mvn clean compile assembly:single
java -jar target/fichero.jar
```

5 Integración, pruebas y resultados

A continuación, se muestran las pruebas que se han llevado a cabo para comprobar el correcto funcionamiento de la arquitectura. El objetivo es twittear una serie de frases que podrían calificarse como posibles frases terroristas para comprobar si la aplicación es capaz de detectarlo. Una vez detectadas las frases, se crearán distintas graficas en Kibana para mostrar los resultados.

5.1 Especificaciones del entorno y de la arquitectura

Las pruebas se han realizado desde un solo ordenador con las siguientes características:

- Hardware: Lenovo Thinkpad T470s, Intel® Core™ i5-7200U CPU @ 2.50Ghz x 4
- Memoria RAM: 8Gb
- Sistema operativo: Kali Linux (64 bits)
- Kafka 2.11-011.0.0
- Standford CoreNLP versión 3.8.0
- Java openjdk 1.8.0_171
- Kibana 6.1.1
- ElasticSearch 6.1.1

5.2 Diccionario de etiquetas

El diccionario utilizado ha sido uno de los que proporciona CoreNLP. Se ha utilizado este diccionario por su alta velocidad de respuesta y su pequeño porcentaje de fallo.

El etiquetador es: edu/stanford/nlp/models/pos-tagger/english-left3words/english-left3words-distsim.tagger

5.3 Orden de ejecución de las herramientas

A la hora de desplegar las herramientas es importante el orden que se debe seguir. Por ejemplo, si ejecutamos primero el programa que lee información de Twitter y lo envía a través de una cola kafka sin haber desplegado anteriormente Apache Kafka, se producirá una excepción. Por lo que el orden de despliegue de las herramientas es el siguiente.

1. Se arranca el servidor de Zookeeper con su fichero de configuración.
2. Se arranca el servidor de Kafka con su fichero de configuración.
3. Se despliegan los 3 componentes de ELK stack independientemente del orden.
4. Se ejecutan los ficheros JAR creados anteriormente para desplegar las aplicaciones de Twitter y de CoreNLP.

5.4 Creación de graficas

En la figura 5-1 se muestra el dashboard general que se ha creado desde el cual se pueden ver 5 tipos distintos de graficas. Estas graficas son explicadas a continuación.

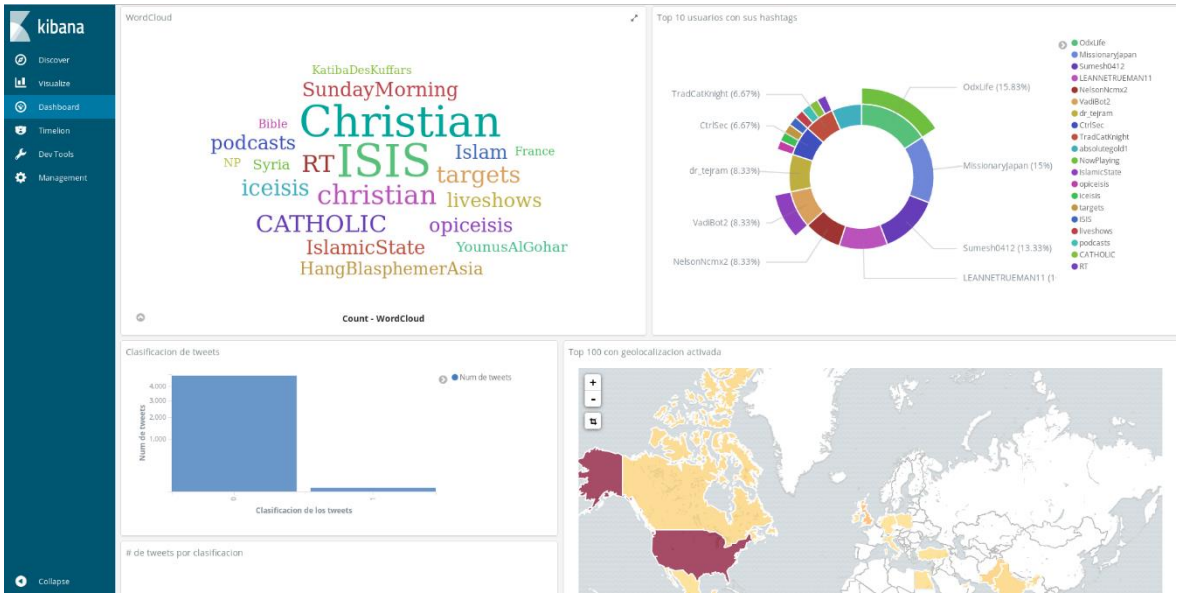

Figura 5-1 Ventana dashboard en Kibana

En la figura 5-2 se muestra una nube de palabras donde se visualizan los *hashtags* que mayor número de veces se repiten, como se puede observar, la mayor parte de ellos tienen que ver con el cristianismo y el islam.

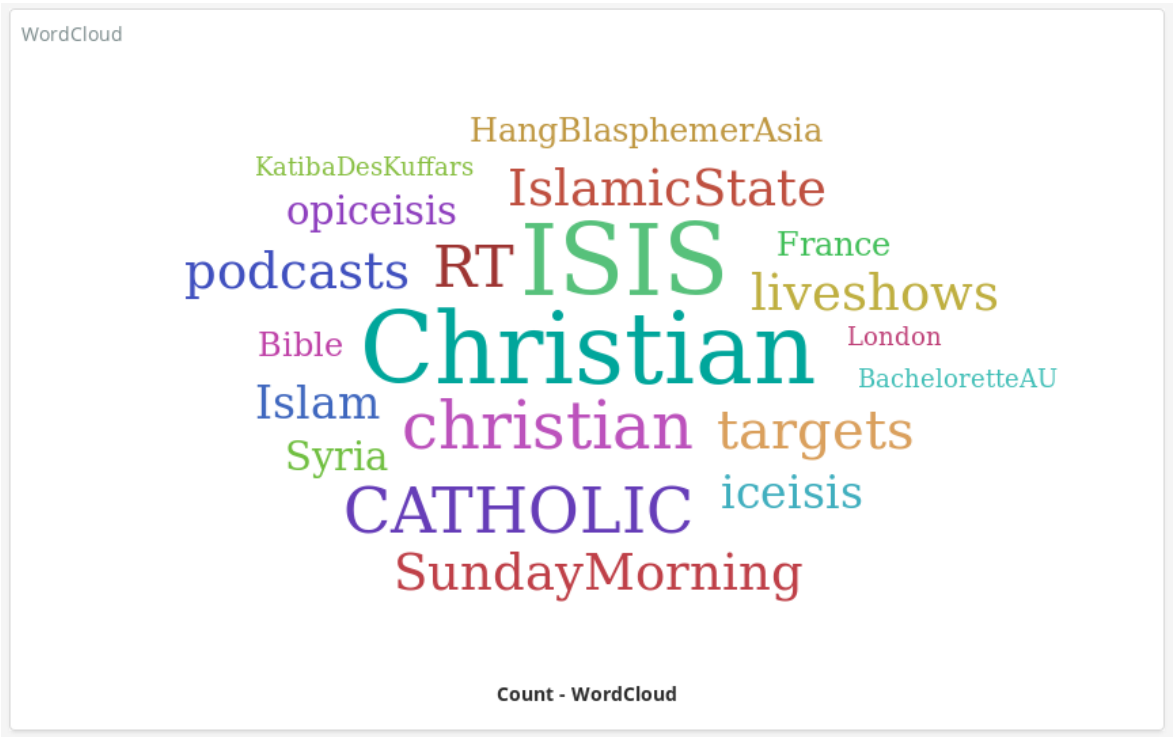

Figura 5-2 Nube de palabras basada en hashtags

En la figura 5-3 se muestran dos visualizaciones que ayudan a entender el número de tweets que han sido clasificados como posibles terroristas respecto a los que no.

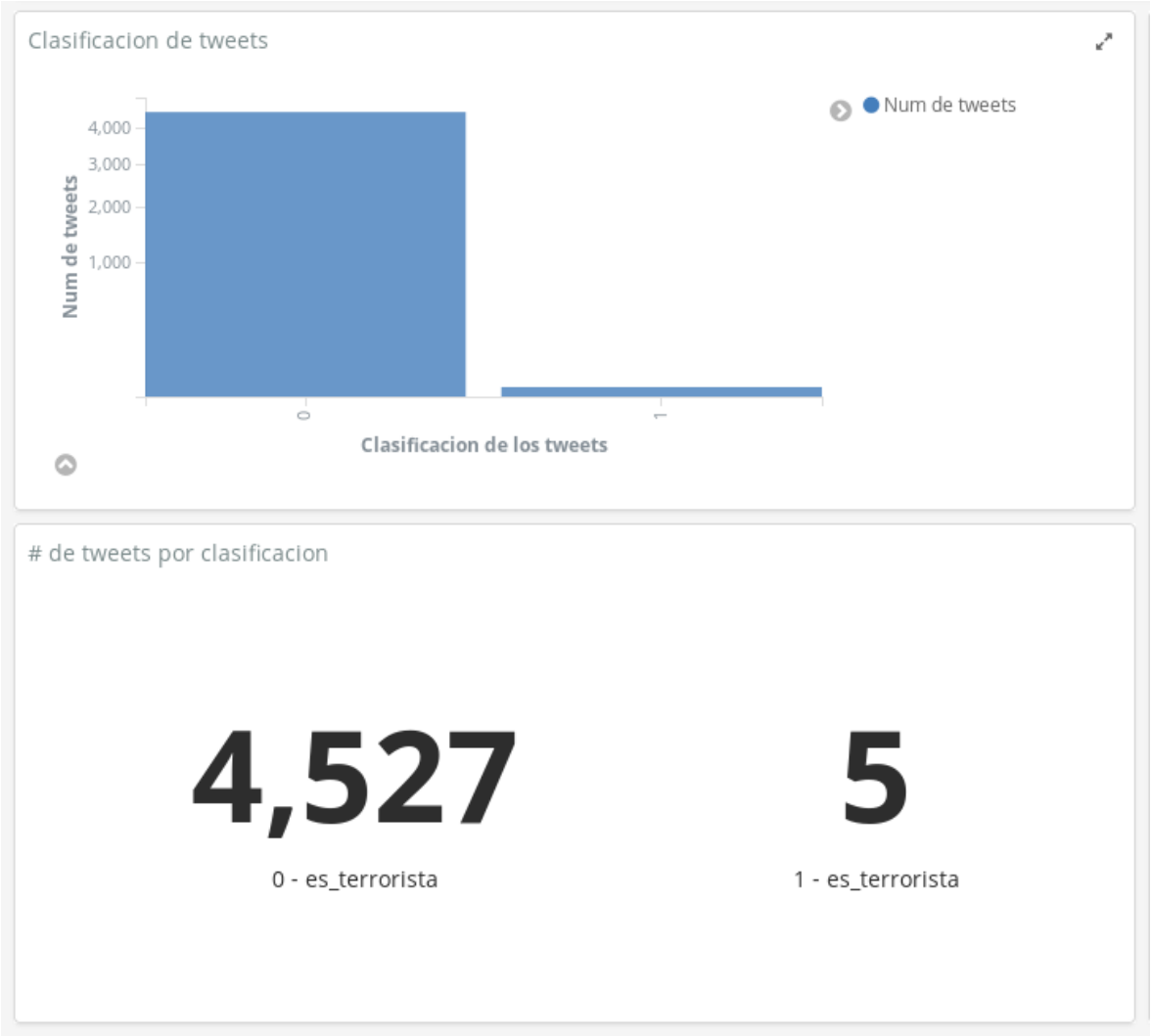

Figura 5-3 Número de tweets según su clasificación

La primera grafica ha tenido que ser ajustada de tal modo que el eje de las X es del modo logarítmico para permitir una mejor visualización del gráfico.

En la figura 5-4 se muestra un gráfico en forma de tarta de los usuarios más activos y cuáles son sus hashtags más utilizados.

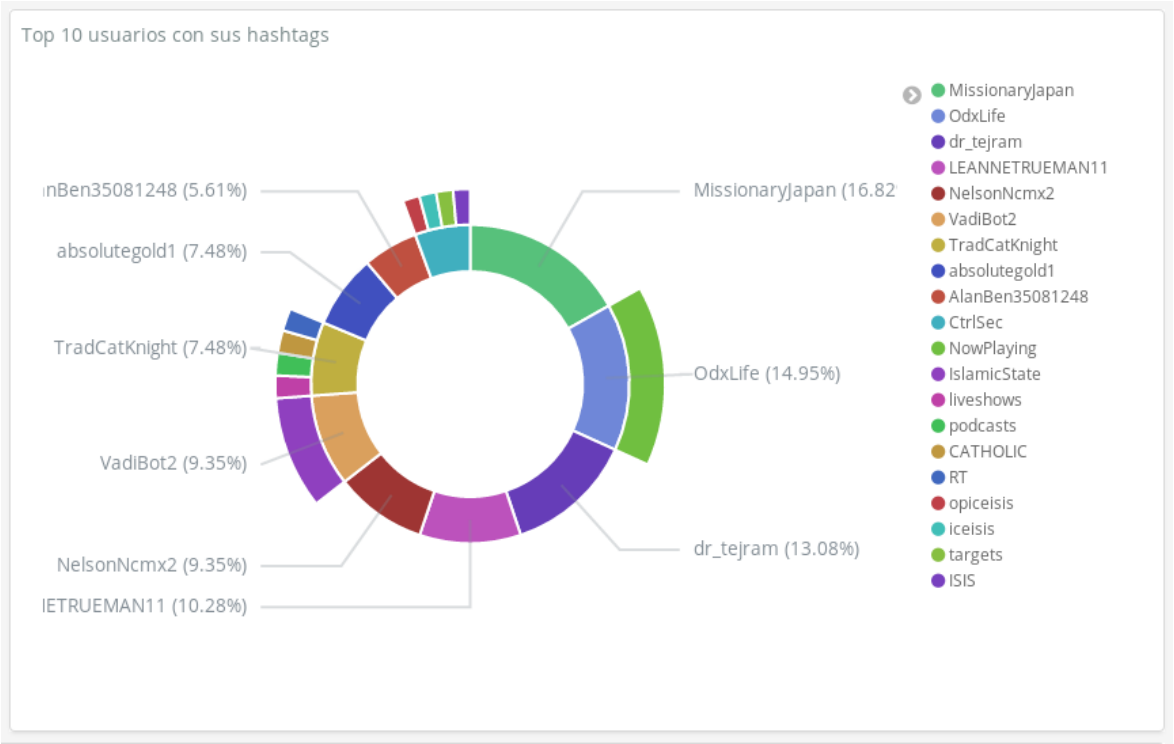

Figura 5-4 Nube de palabras basada en hashtags

Por último, en la figura 5-5 se muestra un mapa de calor geográfico en el que se detallan los países que más tweets han publicado. Cabe destacar que en esta grafica se mostraran únicamente aquellos tweets cuyo usuario tenga habilitada la geolocalización por lo que los resultados no son totalmente exactos.

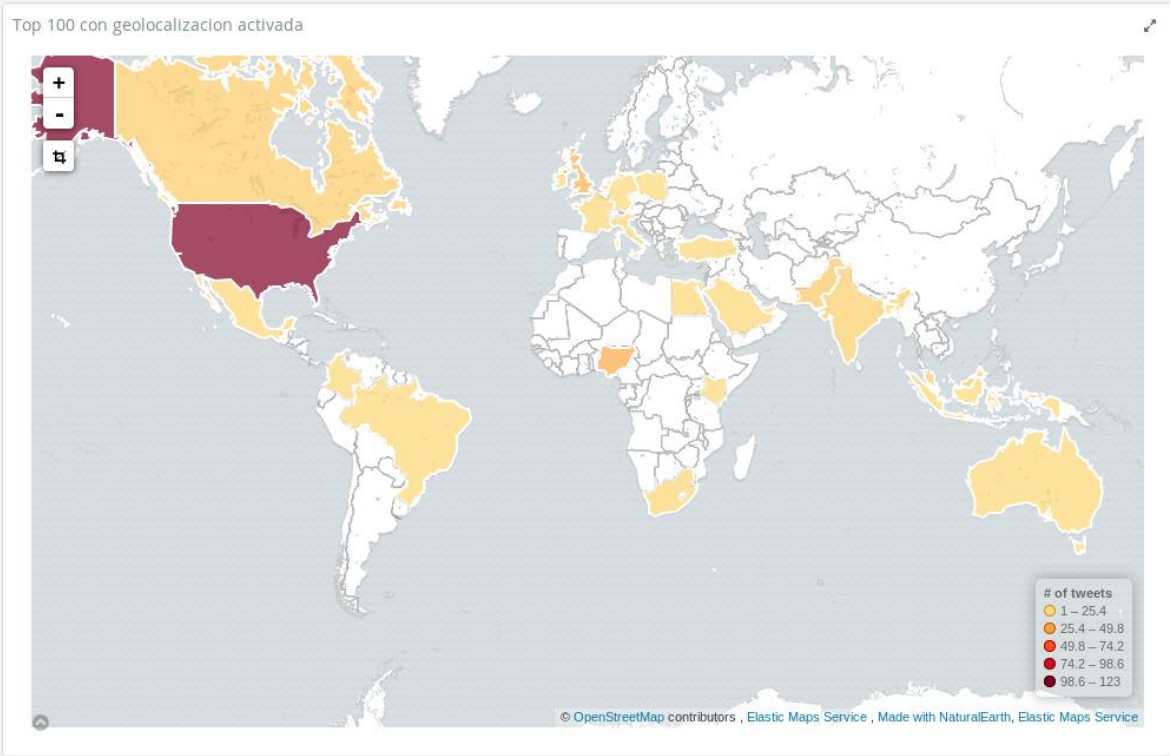

Figura 5-5 Mapa de calor por países

Para la realización de estas gráficas, se han publicado tweets para comprobar el correcto funcionamiento de la herramienta.

A continuación, se muestran algunas de las frases que se han publicado y si han sido detectadas o no por la aplicación.

"The solution is to put some c4 in a church for #IslamicState" → **Detectado**

"I just hope that when the time comes I will be able to actívate my vest in the name of Allah" → **Detectado**

"I will perform an attack to all christians living in UK" → **No detectado** (en este caso *attack* es sustantivo en vez de verbo

"*For allah i will attack all americans in EEUU*" → **Detectado**

"Tired of all these occidental people, we must fight all christians who disrespect the yihad" → **Detectado**

6 Conclusiones y trabajo futuro

6.1 Conclusiones

El objetivo de este Trabajo de Fin de Grado era la creación de una arquitectura que permitiera detectar posibles tweets que dieran lugar al anunciamiento de un atentado futuro. Una vez finalizado el proyecto, se pueden concluir lo siguiente:

- La herramienta es capaz de detectar en un intervalo muy pequeño y razonable tweets categorizados como posibles terroristas. Este intervalo oscila entre 0 y 15 segundos desde que se publica el tweet hasta que es obtenido y procesado.

- La herramienta se comporta bien cuando la ingesta de tweets es elevada en las horas punta.

- Ha sido una buena elección las herramientas utilizadas por los siguientes motivos:
    - Twitter sigue siendo la red social donde más mensajes se comparten.
    - El uso de Kafka proporciona una capa de modularidad perfecto para el proyecto.
    - CoreNLP tiene soporte para el árabe aparte del inglés.
    - ELK está tan bien integrado que su despliegue se puede hacer con un par de líneas de código.

- Aunque la herramienta sea capaz de detectar determinados tweets, es necesario utilizar el criterio humano para descartar falsos positivos.

Todas las anteriores tecnologías han respondido perfectamente a los requisitos que tenía que cumplir el proyecto. Algunos de estos requisitos son la velocidad de procesado o el porcentaje de fallos frente aciertos.

Además, la arquitectura es más ligera de lo que se creía en un principio y ha sido posible desplegar toda ella en un único sistema con un procesador Intel Core i5 y 8Gb de memoria RAM.

6.2 Trabajo futuro

En cuanto a la arquitectura, se propone como trabajo futuro la implementación de nuevos módulos de ingesta de datos. Estos pueden ser desde redes sociales distintas a Twitter como Telegram, Instagram hasta foros yihadistas o comentarios en periódicos que apoyen los atentados.

También sería interesante crear un listado de identificadores de fotos yihadistas que se comparten en Twitter para poder realizar un seguimiento de que usuarios están compartiendo estas imágenes y con qué intención.

Con respecto al procesador de lenguaje, se podría trabajar conjuntamente con una persona que sepa árabe para poder crear reglas y patrones que reconozcan mensajes en árabe. También podrían añadirse más pares de palabras a los *datasets* existentes.

En cuanto a la visualización de los datos, se podría añadir un filtro en logstash para poder visualizar las coordenadas desde las cuales se ponen los tweets ya que, aunque Twitter devuelve un campo con coordenadas, ElasticSearch lo interpreta como un dato numérico entero por lo que no se puede representar en un mapa.

Por último, se propone utilizar *dockers* para automatizar el despliegue de las aplicaciones en los nodos que se quiera utilizar ya que, al tratarse de varias herramientas las que tienen que ejecutarse, el despliegue se podría hacer automáticamente.

Referencias

[1] Risk-track http://risk-track.eu/es/

[2] Jiménez, Ana. "*El yihadismo navega en las redes sociales*". Publicación de septiembre de 2014 en internet.

[3] Mejía Llano, Juan Carlos. "*Estadísticas de redes sociales julio 2018: Usuarios de Facebook, Twitter, Instagram, YouTube, LinkedIn, Whatsapp y otros*". Publicación de noviembre de 2018 en internet.

[4] Espinar, Daniel. "*Nivel morfológico. La estructura de la palabra*". Publicación de octubre de 2011 en internet.

[5] Gupta, Yuvraj. "*Kibana Essentials"*. ISBN 978-1-784-39493-6

[6] García, Samuel. "*NoSQL y los motores de búsqueda: Apache Solr vs Elasticsearch*", Publicación de enero de 2018 en internet.

[7] https://nlp.stanford.edu/software/pos-tagger-faq.html

[8] http://data.cervantesvirtual.com/blog/2017/07/17/libreria-corenlp-de-stanford-de-procesamiento-lenguage-natural-reconocimiento-entidades/

Glosario de abreviaturas

API	*Application Programming Interface*
CPU	*Central processing unit*
DSL	*Domain Specific Languague*
EI	Estado Islámico
ELK	Abreviatura para describir el conjunto de herramientas ElasticSearch, Logstash y Kibana.
FIFO	*First in First out*
GATE	*General Architecture for Text Engineering*
IS	*Islamic State*
ISIL	*Islamic State of Iraq and the Levant*
ISIS	*Islamic State of Iraq and Syria*
JSON	*JavaScript Object Notation*
NLP	*Natural Languaje Processing*
NLTK	*Natural Language Toolkit*
POS	*Part of speech*

Anexos

A Ejemplo de una respuesta de Twitter

A continuación, se muestra un ejemplo de fichero JSON en el que se muestra como es un tweet devuelto por Twitter.


```
{
  "created_at": "Tue Sep 26 21:00:22 +0000 2017",
  "id": 912783930431905797,
  "id_str": "912783930431905797",
  "text": "Can\u2019t fit your Tweet into 140 characters?
\ud83e\udd14\n\nWe\u2019re trying something new with a small group, and
increasing the char\u2026 https:\/\/t.co\/y1rJlHsVB5",
  "source": "<a href=\"http:\/\/twitter.com\" rel=\"nofollow\">Twitter Web
Client<\/a>",
  "truncated": true,
  "in_reply_to_status_id": null,
  "in_reply_to_status_id_str": null,
  "in_reply_to_user_id": null,
  "in_reply_to_user_id_str": null,
  "in_reply_to_screen_name": null,
  "user": {
    "id": 783214,
    "id_str": "783214",
    "name": "Twitter",
    "screen_name": "Twitter",
    "location": "San Francisco, CA",
    "url": "https:\/\/blog.twitter.com\/",
    "description": "Your official source for what\u2019s happening.\n\nNeed a
hand? Visit https:\/\/support.twitter.com",
    "translator_type": "regular",
    "derived": {
      "locations": [
        {
          "country": "United States",
          "country_code": "US",
          "locality": "San Francisco",
          "region": "California",
          "sub_region": "San Francisco County",
          "full_name": "San Francisco, California, United States",
          "geo": {
            "coordinates": [
              -122.41942,
              37.77493
            ],
            "type": "point"
          }
        }
      ]
    },
    "protected": false,
    "verified": true,
```

```
    "followers_count": 61984546,
    "friends_count": 167,
    "listed_count": 90945,
    "favourites_count": 5165,
    "statuses_count": 6148,
    "created_at": "Tue Feb 20 14:35:54 +0000 2007",
    "utc_offset": -25200,
    "time_zone": null,
    "geo_enabled": true,
    "lang": "en",
    "contributors_enabled": false,
    "is_translator": false,
    "profile_background_color": "ACDED6",
    "profile_background_image_url":
"http:\/\/pbs.twimg.com\/profile_background_images\/657090062\/l1uqey5sy82r9ijh
ke1i.png",
    "profile_background_image_url_https":
"https:\/\/pbs.twimg.com\/profile_background_images\/657090062\/l1uqey5sy82r9ij
hke1i.png",
    "profile_background_tile": true,
    "profile_link_color": "1B95E0",
    "profile_sidebar_border_color": "FFFFFF",
    "profile_sidebar_fill_color": "F6F6F6",
    "profile_text_color": "333333",
    "profile_use_background_image": true,
    "profile_image_url":
"http:\/\/pbs.twimg.com\/profile_images\/875087697177567232\/Qfy0kRIP_normal.jp
g",
    "profile_image_url_https":
"https:\/\/pbs.twimg.com\/profile_images\/875087697177567232\/Qfy0kRIP_normal.j
pg",
    "profile_banner_url":
"https:\/\/pbs.twimg.com\/profile_banners\/783214\/1503522865",
    "default_profile": false,
    "default_profile_image": false,
    "following": null,
    "follow_request_sent": null,
    "notifications": null
  },
  "geo": null,
  "coordinates": null,
  "place": null,
  "contributors": null,
  "is_quote_status": false,
  "extended_tweet": {
    "full_text": "Can\u2019t fit your Tweet into 140 characters?
\ud83e\udd14\n\nWe\u2019re trying something new with a small group, and
increasing the character limit to 280! Excited about the possibilities? Read
our blog to find out how it all adds up.
\ud83d\udc47\nhttps:\/\/t.co\/C6hjsB9nbL",
    "display_text_range": [
      0,
      239
    ],
    "entities": {
      "hashtags": [

      ],
      "urls": [
```

```
        {
          "url": "https:\/\/t.co\/C6hjsB9nbL",
          "expanded_url": "https:\/\/cards.twitter.com\/cards\/gsby\/4ubsj",
          "display_url": "cards.twitter.com\/cards\/gsby\/4ub\u2026",
          "unwound": {
            "url": "https:\/\/cards.twitter.com\/cards\/gsby\/4ubsj",
            "status": 200,
            "title": "Giving you more characters to express yourself",
            "description": null
          },
          "indices": [
            216,
            239
          ]
        }
      ],
      "user_mentions": [

      ],
      "symbols": [

      ]
    }
  },
  "quote_count": 0,
  "reply_count": 16027,
  "retweet_count": 47906,
  "favorite_count": 78829,
  "entities": {
    "hashtags": [

    ],
    "urls": [
      {
        "url": "https:\/\/t.co\/y1rJlHsVB5",
        "expanded_url":
"https:\/\/twitter.com\/i\/web\/status\/912783930431905797",
        "display_url": "twitter.com\/i\/web\/status\/9\u2026",
        "indices": [
          117,
          140
        ]
      }
    ],
    "user_mentions": [

    ],
    "symbols": [

    ]
  },
  "favorited": false,
  "retweeted": false,
  "possibly_sensitive": false,
  "filter_level": "low",
  "lang": "en",
  "matching_rules": [
    {
      "tag": null
```

```
    }
  ]
}
```

B Ficheros Maven

A continuación, se muestran los ficheros Maven necesarios para la compilación de las clases java en un fichero JAR.

B.1 pom.xml del módulo de Twitter

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>TwitterConsumer</groupId>
    <artifactId>TwitterConsumer</artifactId>
    <version>1.0</version>

    <licenses>
        <license>
            <name>Apache License 2.0</name>
            <url>http://www.apache.org/licenses/LICENSE-2.0</url>
            <distribution>repo</distribution>
        </license>
    </licenses>

    <developers>
        <developer>
            <name>Alvaro Vadillo</name>
            <email>alvaro.vadillo@estudiante.uam.es</email>
            <url>github.com/AlvaroVadillo</url>
            <roles>
                <role>lead</role>
                <role>architect</role>
                <role>developer</role>
            </roles>
        </developer>
    </developers>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>Main</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

```
    <dependencies>
        <dependency>
            <groupId>org.twitter4j</groupId>
            <artifactId>twitter4j-core</artifactId>
            <version>4.0.6</version>
        </dependency>
        <dependency>
            <groupId>org.twitter4j</groupId>
            <artifactId>twitter4j-async</artifactId>
            <version>4.0.6</version>
        </dependency>
        <dependency>
            <groupId>org.twitter4j</groupId>
            <artifactId>twitter4j-stream</artifactId>
            <version>4.0.6</version>
        </dependency>
        <dependency>
            <groupId>org.twitter4j</groupId>
            <artifactId>twitter4j-media-support</artifactId>
            <version>4.0.6</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.11.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka_2.11</artifactId>
            <version>0.11.0.0</version>
        </dependency>
    </dependencies>
</project>
```

B.2 pom.xml del módulo de CoreNLP

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>CoreNLP</groupId>
    <artifactId>CoreNLP</artifactId>
    <version>1.0-SNAPSHOT</version>

    <licenses>
        <license>
            <name>Apache License 2.0</name>
            <url>http://www.apache.org/licenses/LICENSE-2.0</url>
            <distribution>repo</distribution>
        </license>
    </licenses>

    <developers>
        <developer>
            <name>Alvaro Vadillo</name>
            <email>alvaro.vadillo@estudiante.uam.es</email>
            <url>github.com/AlvaroVadillo</url>
            <roles>
                <role>lead</role>
                <role>architect</role>
                <role>developer</role>
            </roles>
        </developer>
    </developers>


    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>Main</mainClass>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>edu.stanford.nlp</groupId>
            <artifactId>stanford-corenlp</artifactId>
```

```
            <version>3.8.0</version>
        </dependency>
        <dependency>
            <groupId>edu.stanford.nlp</groupId>
            <artifactId>stanford-corenlp</artifactId>
            <version>3.8.0</version>
            <classifier>models</classifier>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.11.0.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka_2.11</artifactId>
            <version>0.11.0.0</version>
        </dependency>
        <!--Use for json twitter library*/-->
        <dependency>
            <groupId>org.twitter4j</groupId>
            <artifactId>twitter4j-core</artifactId>
            <version>4.0.6</version>
        </dependency>
    </dependencies>
</project>
```

C Listado de objetos directos utilizados

A continuación, se muestra el listado de objetos directos (dobj) utilizados en el proyecto:

```
dobj(set, bomb)
dobj(set, bombs)
dobj(set, explosive)
dobj(set, explosives)
dobj(set, artifact)
dobj(set, artifacts)
dobj(set, mine)
dobj(set, mines)
dobj(set, c4)
dobj(put, bomb)
dobj(put, bombs)
dobj(put, explosive)
dobj(put, explosives)
dobj(put, artifact)
dobj(put, artifacts)
dobj(put, mine)
dobj(put, mines)
dobj(put, c4)
dobj(explode, bomb)
dobj(explode, bombs)
dobj(explode, explosive)
dobj(explode, explosives)
dobj(explode, artifact)
dobj(explode, artifacts)
dobj(explode, mine)
dobj(explode, mines)
dobj(explode, c4)
dobj(explode, vest)
dobj(explode, vests)
dobj(blast, bomb)
dobj(blast, bombs)
dobj(blast, explosive)
dobj(blast, explosives)
dobj(blast, artifact)
dobj(blast, artifacts)
dobj(blast, mine)
dobj(blast, mines)
dobj(blast, c4)
dobj(blast, vest)
dobj(blast, vests)
dobj(activate, bomb)
dobj(activate, bombs)
dobj(activate, explosive)
dobj(activate, explosives)
dobj(activate, artifact)
dobj(activate, artifacts)
dobj(activate, c4)
dobj(activate, vest)
dobj(activate, vests)
dobj(kill, people)
dobj(kill, person)
dobj(kill, infidel)
dobj(kill, infidels)
```

dobj(kill, christian)
dobj(kill, christians)
dobj(kill, occidental)
dobj(kill, occidentals)
dobj(kill, apostate)
dobj(kill, apostates)
dobj(kill, european)
dobj(kill, europeans)
dobj(kill, american)
dobj(kill, americans)
dobj(murder, people)
dobj(murder, person)
dobj(murder, infidel)
dobj(murder, infidels)
dobj(murder, christian)
dobj(murder, christians)
dobj(murder, occidental)
dobj(murder, occidentals)
dobj(murder, apostate)
dobj(murder, apostates)
dobj(murder, european)
dobj(murder, europeans)
dobj(murder, american)
dobj(murder, americans)
dobj(fight, infidel)
dobj(fight, infidels)
dobj(fight, christian)
dobj(fight, christians)
dobj(fight, occidental)
dobj(fight, occidentals)
dobj(fight, apostate)
dobj(fight, apostates)
dobj(fight, european)
dobj(fight, europeans)
dobj(fight, american)
dobj(fight, americans)
dobj(burn, infidel)
dobj(burn, infidels)
dobj(burn, christian)
dobj(burn, christians)
dobj(burn, occidental)
dobj(burn, occidentals)
dobj(burn, apostate)
dobj(burn, apostates)
dobj(burn, european)
dobj(burn, europeans)
dobj(burn, american)
dobj(burn, americans)
dobj(attack, infidel)
dobj(attack, infidels)
dobj(attack, christian)
dobj(attack, christians)
dobj(attack, occidental)
dobj(attack, occidentals)
dobj(attack, apostate)
dobj(attack, apostates)
dobj(attack, european)
dobj(attack, europeans)
dobj(attack, american)

dobj(attack, americans)
dobj(suffer, infidel)
dobj(suffer, infidels)
dobj(suffer, christian)
dobj(suffer, christians)
dobj(suffer, occidental)
dobj(suffer, occidentals)
dobj(suffer, apostate)
dobj(suffer, apostates)
dobj(suffer, european)
dobj(suffer, europeans)
dobj(suffer, american)
dobj(suffer, americans)